



XTCE

XML Telemetry & Command Exchange Tutorial, XTCE Version 1.1

Kevin Rice, NASA GSFC, GST Inc., Kevin.Rice@gst.com

Brad Kizzort, Harris Corp., bkizzort@harris.com

March 23, 2008

Contents



- Preface
 - Brief history and background
 - Applicability
 - XML and XML Schema Basics
 - A little UML
- Chapter 1
 - Overview of major XTCE Schema areas
- Chapter 2
 - Describing a telemetry item
- Chapter 3
 - Grouping items into a package description
 - Packets, Minor Frames, etc...
- Chapter 4
 - Command basics in XTCE
- Chapter 5
 - The Forgotten Elements
- Chapter 6
 - Implementing XTCE, natively or for exchange
 - Issues
- Summary

Preface - Brief History and Background



- XTCE 1.0 first published in 2005
 - OMG's Space Domain Task Force (SDTF) – from 2000
 - { ESA, Boeing Comm. Sat., US AF SMC Det 12/VO – Lockheed Martin } original submitters + NASA/GSFC + Harris Corp.
- CCSDS Published XTCE 1.1 in Oct 2007 as Blue Book
 - XTCE 1.1 revved in conjunction w/CCSDS agency review
 - XTCE 1.1 Green Book (overview) 2007
 - XTCE 1.1 Magenta Books (user guide/ccsds tailoring - out or draft available)
 - XTCE 1.2 – early drafts to clean up some minor bugs... nothing official yet
- Goal
 - Provide industry w/standard mechanism for describing telemetry and command streams (principally from satellites)
 - Lower cost and increase validation over traditional formats
 - XML widely adopted by industry, many technologies available to leverage
 - XML Schema provides some validation
 - Support exchange or native format



Preface - Applicability

- XTCE is designed to describe bit-streams
 - “bit-packed” - no markers, or metadata, non-self describing
 - Typical of telemetry & command in historic space domain
 - Focus is really at the packet or minor frame level
 - Although there is an element for “Stream” descriptions
 - For example, CCSDS packet descriptions, telemetry and command
 - Probably not the CCSDS frame, RS, CADU, etc..., (FEP) – outside of XTCE
- Use for Exchange, but...
 - Best in a team scenario where all parties are implementing the same format
 - As this separation increases, 100% exchange is less and less achievable
- Or native format
 - Nothing precludes using XTCE as the native format for toolset that supports to decommutate telemetry or build commands



Preface – Quick XML

- XML

<Tag>

<SubTag attribute="Data1"/> <!-- comment: tags are also called elements -->

Data2

</Tag>

- XML Schema

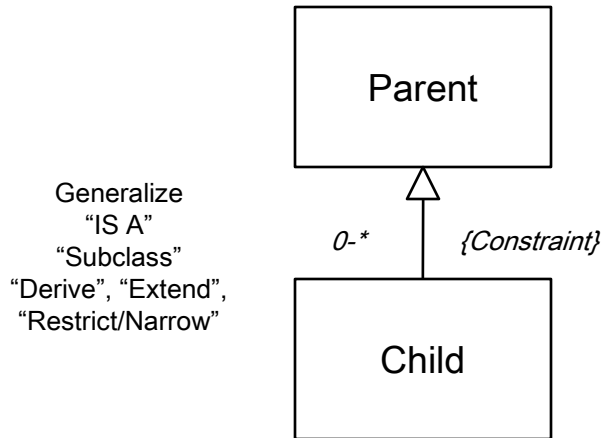
- Rules for defining your tags and attributes
 - Order, optional or not, choice of
- And their data-types, value ranges
- And their cardinality (count)
- Validate XML files with your XML Schema using an XML parser

- XTCE is an XML Schema

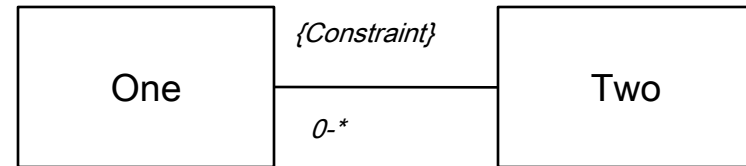
- Create specific XTCE XML files describing your telemetry and command format
- Validate XTCE XML files with the XTCE Schema
 - XTCE XML files – aka "Instance documents"
- Exchange: give your XTCE file to your friend



Preface - Quick UML



Child Inherits properties from Parent
-- OR overrides properties explicit in the
Parent



Association
"HAS A"

Note: various styles of association are present in UML
-Aggregation
-Composition
-Bidirectional/unidirectional

One "asks" Two to do something

XTCE has an Object Oriented Model in its Container and Command areas
Basic UML Diagrams to describe the relationships present in the examples

Chapter 1 - XTCE Overview



```
<SpaceSystem>  
  <TelemetryMetaData/>  
  <CommandMetaData/>  
  <SpaceSystem/>  
</SpaceSystem>
```

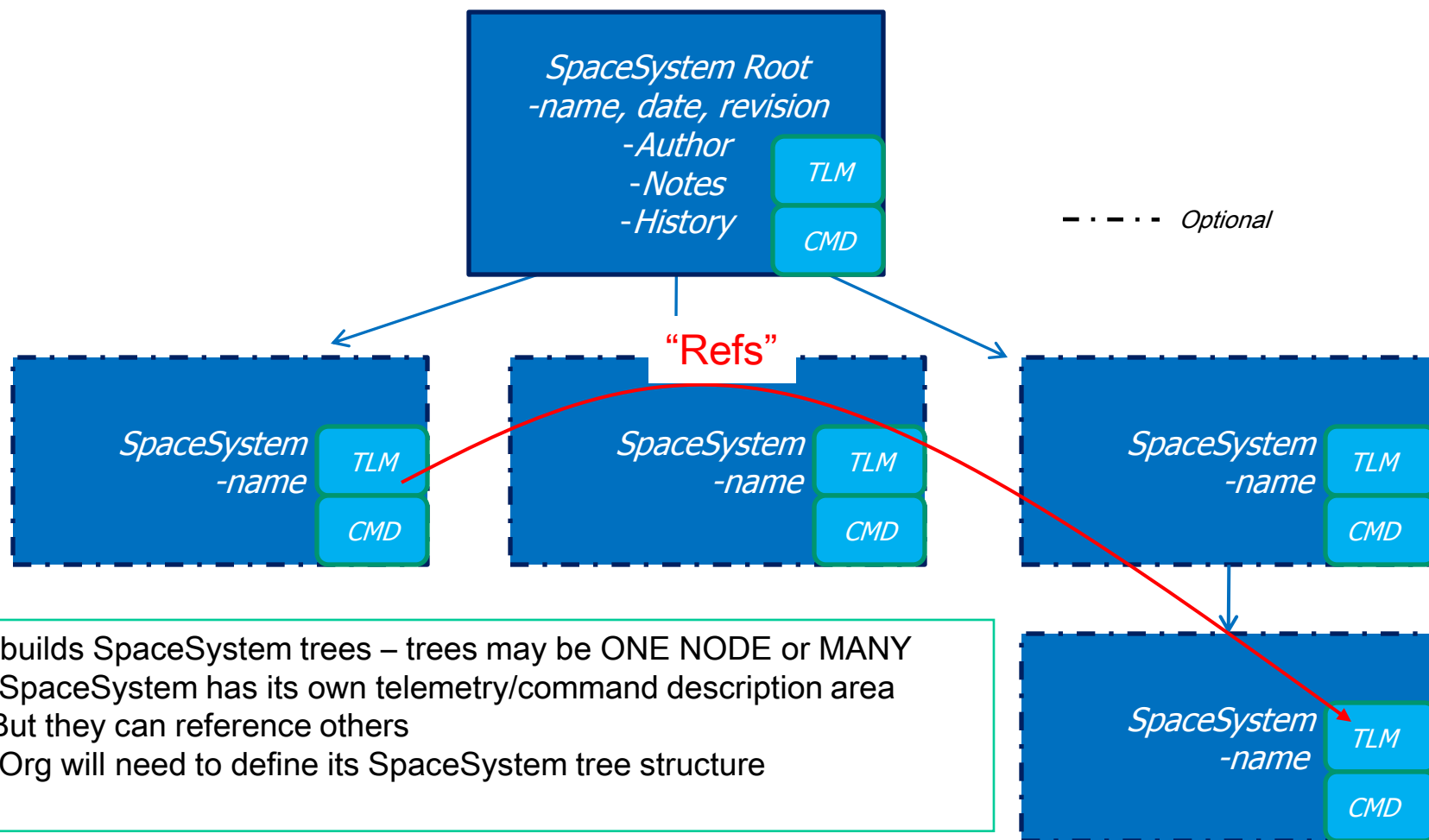
Chapter 1 – XTCE Overview (Cont'd)



```
<SpaceSystem>  
  <TelemetryMetaData/>  
  <CommandMetaData/>  
  <SpaceSystem/>  
</SpaceSystem>
```

- Space System
 - Element: **<SpaceSystem>**
 - one or many in a tree configuration
 - Optional child elements for things like date, authors, etc...

Chapter 1 - XTCE SpaceSystem Tree(s)



- XTCE builds SpaceSystem trees – trees may be ONE NODE or MANY
- Each SpaceSystem has its own telemetry/command description area
 - But they can reference others
 - Each Org will need to define its SpaceSystem tree structure



Chapter 1 – XTCE Overview (Cont'd)

```
<SpaceSystem>
  <TelemetryMetaData>
    <ParameterSet/>
    <ParameterTypeSet>
  </TelemetryMetaData>
  <CommandMetaData/>
  <SpaceSystem/>
</SpaceSystem>
```

- Telemetry Items

- Elements: **<Parameter>** & **<ParameterTypes>** “Parameters”
- Link Data Type of link info (e.g. integer count)
 - Integer, Float, String, Binary
 - Size in Bits, format (IEEE784, twos complement, etc...), signed/unsigned, bit order, byte order, etc...
- Host Data Type after conversion to host (e.g. float)
 - Enumerated, String, Binary, Integer, Float, Boolean, Time (absolute/relative), array, aggregate (“struct”)
 - Size in bits, initial value, signed/unsigned, etc...
- Range of Content (integer/float)
- Calibration (varies by ParameterType)
- Alarms (also varies by ParameterType)



Chapter 1 - SpaceSystem Diagram in XTCE

```
<xtce:SpaceSystem name="MyRoot">
  <xtce:TelemetryMetaData/>
  <xtce:CommandMetaData/>
  <xtce:SpaceSystem name="MySubSys1"/>
  <xtce:SpaceSystem name="MySubSys2"/>
  <xtce:SpaceSystem name="MySubSys3">
    <xtce:TelemetryMetaData/>
    <xtce:CommandMetaData/>
    <xtce:SpaceSystem name="MySubSubSys3.1"/>
  <xtce:SpaceSystem>
</xtce:SpaceSystem>
```

← Common Tlm/Cmd

← Tlm/Cmd for this sub-sys

Note: child elements of SpaceSystem not shown for purposes of illustration



Chapter 1 – XTCE Overview (Cont'd)

```
<TelemetryMetaData>  
  <ParameterSet/>  
  <ParameterTypeSet/>  
  <ContainerSet>  
    <SequenceContainer/>  
  </ContainerSet>  
</TelemetryMetaData>
```

- Telemetry Packaging (packets, minor frames)
 - Element: **<SequenceContainer>** “Containers”
 - Specify Sequence of parameters or other containers in the packet/minor frame
 - Optional include condition, addressing and repeat (super-sampling)
 - Size in bits – optional – default: calculated from entries
 - Optional rate of update, bit/byte order, etc...
 - Optionally **EXTEND** another container
 - Element: **<BaseContainer>** and its child **<RestrictionCriteria>** (constraint)
 - Provide name of other container and constraint expression
 - Child inherits certain properties from parent, mainly its entries
 - Use to provide identifying information of packet/minor frame



Chapter 1 – XTCE Overview (Cont'd)

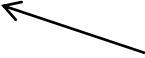
- Commanding: element **<CommandMetaData>**
 - Command Parameter and ParameterType
 - Same construction as on telemetry side - ignore alarms
 - Interpreted as host to uplink
 - System supplied information (checksum, etc...)
 - Command Description
 - Element: **<MetaCommand>**
 - Describe command, its arguments, local packaging information verifiers and other related items
 - Child -- **Local Packaging** is called: **<CommandContainer> (command packet)**
 - Child Element: **<Argument>** links to **<ArgumentType>**
 - Similar underlying construction Parameter/ParameterType
 - Interpreted the same way Command Parameters are interpreted
 - Meant to be used for user supplied information (command argument)
 - Command Container **<CommandContainerSet>**
 - Element: **<CommandContainer>**
 - Similar to SequenceContainer, refer to it in MetaCommand/CommandContainer
 - Group common command parameters reused by more than one command

```
<SpaceSystem>  
  <TelemetryMetaData/>  
  <CommandMetaData>  
    <ParameterSet/>  
    <ParameterTypeSet/>  
    <MetaCommandSet/>  
    <CommandContainerSet/>  
  </CommandMetaData>  
  <SpaceSystem/>  
</SpaceSystem>
```



Chapter 1 – XTCE Overview (Cont'd)

```
<TelemetryMetaData>
  <ParameterTypeSet>
    <IntegerParameterType name="MyType"/>
  </ParameterTypeSet>
  <ParameterSet>
    <Parameter name="MyParam" parameterType="MyType"/>
  </ParameterSet>
</TelemetryMetaData>
```

An arrow points from the right side of the slide to the `parameterType="MyType"` attribute in the `<Parameter>` tag.

- NameReferences

- Many major XTCE elements refer to other areas in an XML file
- XTCE's version of "pointers" are called **NameReferences**
 - Just strings with a specific format: "String Pointers"
- Several Variations
 - Plain & Unix-like path: /SpaceSystemName1/SpaceSystemName2/.../itemName
- Absolute, relative, and the "no path" (plain) version
 - Note: plain is not global; refers to local SpaceSystem first or up "the SpaceSystem tree" relative to that location
- Outside of XML parsing unfortunately – you must implement
- Careful implementation required – used in many XTCE areas:
 - Parameter & ParameterType, Argument and ArgumentType
 - All Containers, MetaCommand, all Comparisons... others?



```
<SpaceSystem>  
  <TelemetryMetaData>  
    <ParameterTypeSet/ >  
    <ParameterSet>  
  </TelemetryMetaData>  
  <CommandMetaData/>  
  <SpaceSystem/>  
</SpaceSystem>
```



<ParameterTypeSet>

<StringParameterType/>

<EnumeratedParameterType/>

<BinaryParameterType/>

<IntegerParameterType/>

<FloatParameterType/>

<BooleanParameterType/>

<AbsoluteTimeParameterType/>

<RelativeTimeParameterType/>

<ArrayParameterType/>

<AggregateParameterType/>

</ParameterTypeSet>

Host Side Data Type ← Link Info

String,

Enumerated,

Binary,

Integer

Float

Boolean

AbsoluteTime

RelativeTime

Array

Aggregate (groups other Parameters)



<StringParameterType>

<StringDataEncoding/>

<IntegerDataEncoding/>

<FloatDataEncoding/>

<BinaryDataEncoding/>

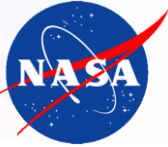
<Alarms/>

*Choice of One
or
None*

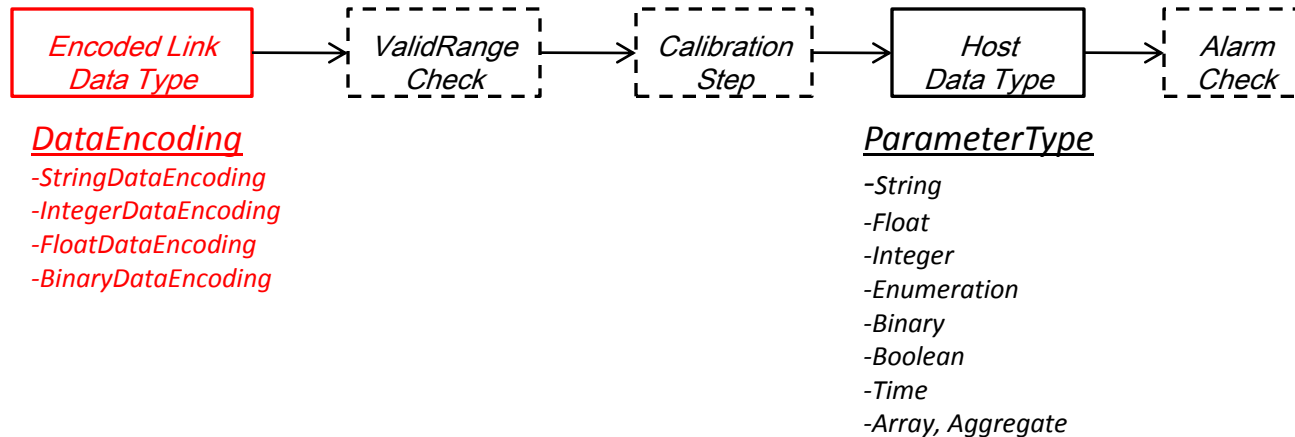
</StringParameterType>

Link Side Data Encoding: how is the information described on the link?

- Choice of one (or none)
- Not all ParameterType & DataEncoding make sense necessarily
- Alarms and some additional specifics vary by ParameterType



- ParameterTypes have the following relationship between information on the “link” and ground
 - and has specific elements and attributes to describe these items



Missions will likely wish to restrict these various elements and attributes for various ParameterType...



Chapter 2 - DataEncodings

- Four options, various details:
 - StringDataEncoding
 - UTF8 or UTF16 encoded Unicode (UTF-16 big endian)
 - Bit size, various ways
 - IntegerDataEncoding
 - Unsigned or TwosComplement (spelled "Compliment" in XTCE!)
 - Bit size
 - bit/byte order
 - Various calibrators such as Linear Calibrator or Polynomial Calibrator optional
 - FloatDataEncoding
 - IEEE-745, MIL1750A
 - Bit size
 - bit/byte order
 - Various calibrators such as Linear Calibrator or Polynomial Calibrator optional
 - BinaryDataEncoding
 - Bit size
 - bit/byte order
 - No cals...

Some additional details left out



Chapter 2 - DataEncoding – Examples

Unsigned int, 32 bits

```
<xtce:IntegerDataEncoding sizeInBits="32"/>
```

Two Complement, 8 bits

```
<xtce:IntegerDataEncoding encoding="twosCompliment"/>
```

Float, 32-bit, IEEE-754

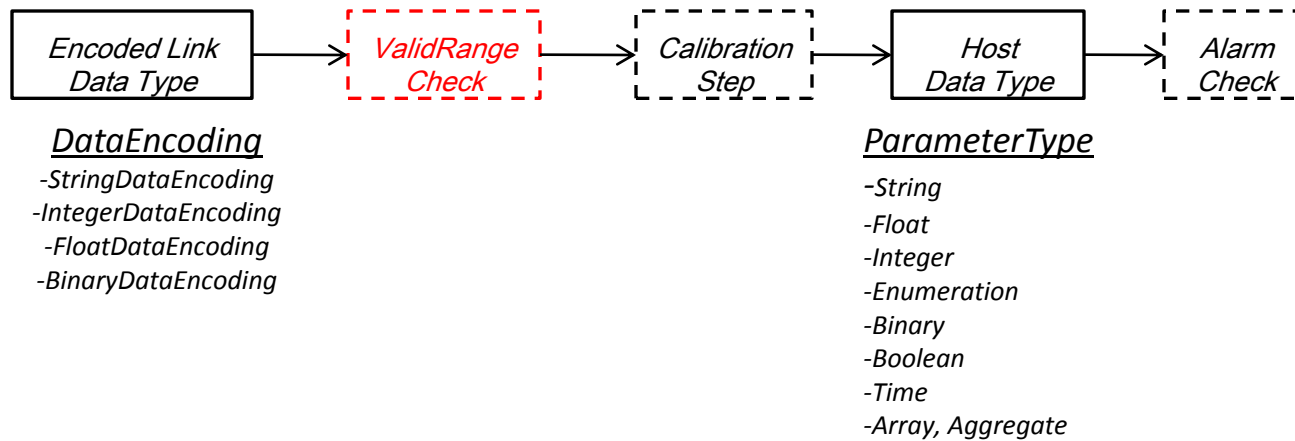
```
<xtce:FloatDataEncoding/>
```

Unicode String, UTF-16

```
<xtce:StringDataEncoding encoding="UTF-16">  
  <!-- Example: Length in bits is 11 characters -->  
  <!-- 16-bits per character -->  
  <xtce:SizeInBits>  
    <xtce:Fixed>  
      <xtce:FixedValue>176</xtce:FixedValue>  
    </xtce:Fixed>  
  </xtce:SizeInBits>  
</xtce:StringDataEncoding>
```

Note: defaults are not shown, the parser should provide it, helps keep the file shorter

Chapter 2 - Telemetry ParameterTypes – Reference Diagram





Chapter 2 - ValidRange Check

- ValidRange Check
 - IntegerParameterType and FloatParameterType only

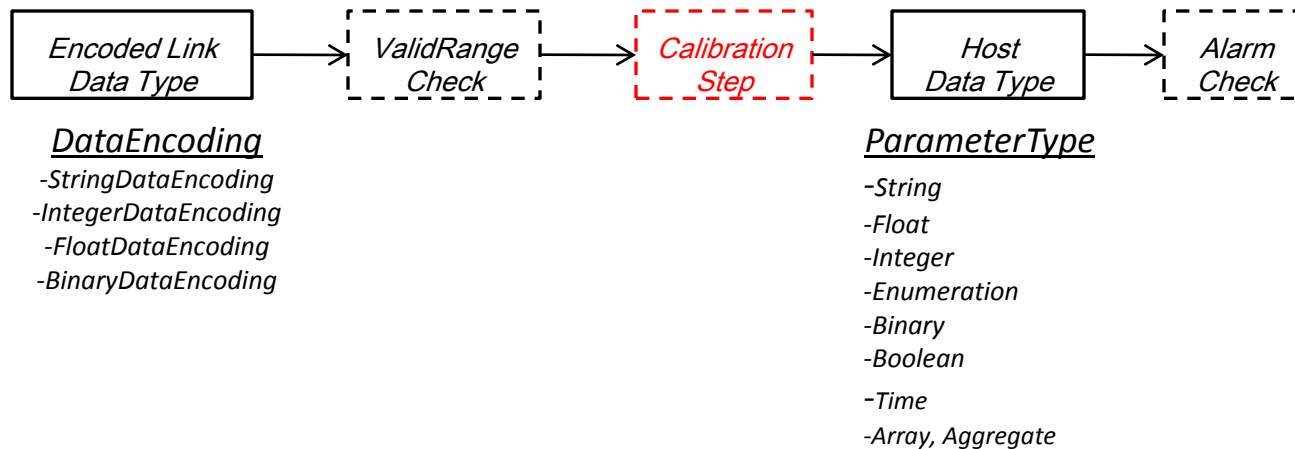
<xtce:ValidRange minInclusive="0" maxInclusive="100"/>

Ignore for Telemetry side:

- IntegerParameterType/@validRangeAppliesToCalibrated=(**true** | false)
- FloatParameterType/@validRangeAppliesToCalibrated=(**true** | false)



Chapter 2 - Telemetry ParameterTypes – Reference Diagram





- Calibrators are defined in the DataEncoding area
- A DefaultCalibrator
 - Just one
- Or ContextCalibrators
 - Many
 - Conditional “Contexts”, user defined
- We’ll look at two common types
 - polynomial (PolynomialCalibrator)
 - line segment (SplineCalibrator)



Chapter 2 - Linear Calibrator

```
<xtce:SplineCalibrator>  
  <!--Forward (regular) calibration -->  
  <xtce:SplinePoint raw="1" calibrated="10"/>  
  <xtce:SplinePoint raw="2" calibrated="100"/>  
  <xtce:SplinePoint raw="3" calibrated="500"/>  
</xtce:SplineCalibrator>
```

There's an optional @order attribute in SplinePoint – this is a typo, ignore

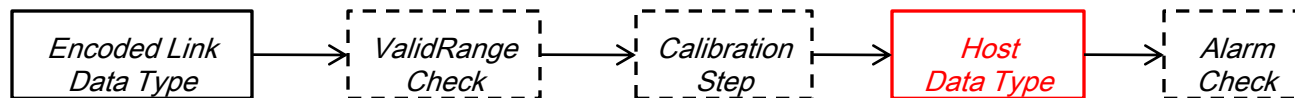


Chapter 2 - Polynomial Calibrator

The equation for the calibration is: $y = -0.0048x^2 + 1.4091x - 48.886$

```
<xtce:PolynomialCalibrator>  
  <!--Forward (regular) calibration -->  
  <xtce:Term exponent="0" coefficient="-48.886"/>      - 48.886  
  <xtce:Term exponent="1" coefficient="1.4091"/>      1.4091x  
  <xtce:Term exponent="2" coefficient="-0.0048"/>      - 0.0048x2  
</xtce:PolynomialCalibrator>
```

Chapter 2 - Telemetry ParameterTypes – Reference Diagram



DataEncoding

- StringDataEncoding
- IntegerDataEncoding
- FloatDataEncoding
- BinaryDataEncoding

ParameterType:

- String,
- Enumerated,
- Binary (catch all),
- Integer
- Float
- Boolean
- AbsoluteTime
- RelativeTime
- Array
- Aggregate (groups of other Parameters)

Chapter 2 – Accepted ParameterType/DataEncoding Combinations



StringParameterType + StringDataEncoding

- Unicode String encoded as either UTF-8 or UTF-16

BooleanParameterType + IntegerDataEncoding

- True/False

EnumeratedParameterType + IntegerDataEncoding

- LABEL, VALUE pairs

BinaryParameterType + BinaryDataEncoding

- Blob data

IntegerParameterType + IntegerDataEncoding

- Integers of various well known formats
 - (one/twos complement, etc...)
- Calibrated/uncalibrated

FloatParameterType + FloatDataEncoding

- Floats of well known formats (IEEE/MIL1750A)
- Calibrated/uncalibrated

FloatParameterType + IntegerDataEncoding

- Integer counts to units conversion
- Calibrated

AbsoluteTimeParameterType + IntegerDataEncoding

- Time
- Same for RelativeTimeParameterType
- Variation for ASCII format

Aggregate and Array ParameterType

- N/A

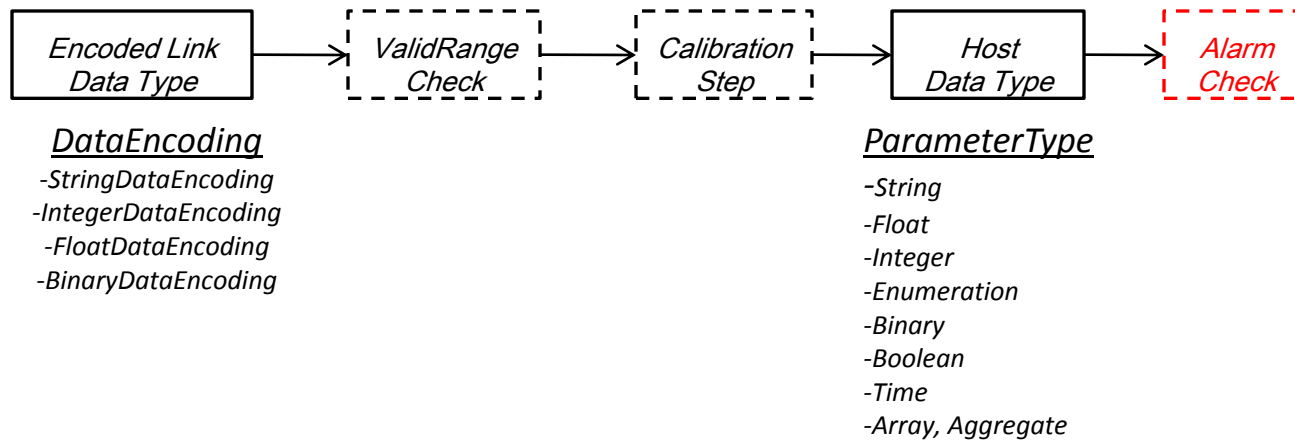
Integer/FloatParameterType + BinaryDataEncoding

- Float/Integer format not in XTCEs default list
- Add Ancillary to help document

Others?

- not accepted but not illegal!

Chapter 2 - Telemetry ParameterTypes – Reference Diagram

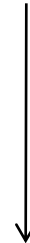




Chapter 2 - Alarms

- DefaultAlarm and ContextAlarms also
- A variety of limit checks are available in XTCE and tuned to their ParameterType somewhat:
 - AlarmConditions: check a condition
 - StaticAlarmRanges: compare values against set of ranges
 - ChangeAlarms: Rate or Delta
 - Slight variations for String & Enum alarms

XTCE Alarm Ranges (optional)
Normal - Inside Least Severe Range
WatchRange
WarningRange
CriticalRange
SevereRange



More severe ranges clip lower ranges

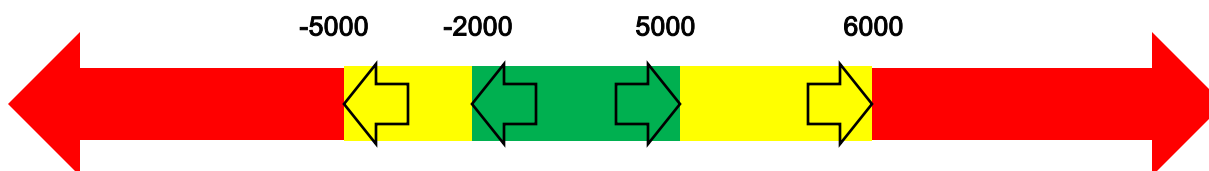
Chapter 2 – StaticAlarmRange Example



```
<xtce:StaticAlarmRanges>  
  <xtce:WatchRange minInclusive="-2000.0" maxInclusive="5000.0"/>  
  <xtce:SevereRange minInclusive="-5000.0" maxInclusive="6000.0"/>  
</xtce:StaticAlarmRanges>
```

Real Ranges:

- Green: $-2000.0 > x < 5000.0$ – implied
- Watch: $x \leq -2000.0$ or $x \geq 5000.0$
- Severe: $x \leq -5000.0$ or $x \geq 6000.0$



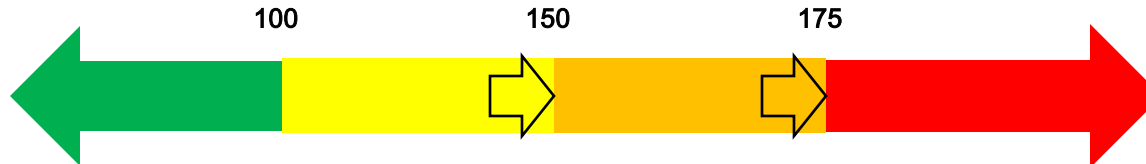
Note: Color designation is not part of XTCE, user dependent



Chapter 2 - AlarmConditions

- Set up conditions for up to five levels
 - The alarm “returns” the highest level triggered
 - Simple single conditions, multiple conditions, boolean expressions, custom algorithms

```
<xtce:AlarmConditions>  
  <xtce:WarningAlarm>  
    <xtce:Comparison parameterRef="pressureValue" value="100" comparisonOperator=">"/>  
  </xtce:WarningAlarm>  
  <xtce:CriticalAlarm>  
    <xtce:Comparison parameterRef="pressureValue" value="150" comparisonOperator=">"/>  
  </xtce:CriticalAlarm>  
  <xtce:SevereAlarm>  
    <xtce:Comparison parameterRef="pressureValue" value="175" comparisonOperator=">"/>  
  </xtce:SevereAlarm>  
</xtce:AlarmConditions>
```





Chapter 2 - ChangeRateAlarms

- Similar to the others in terms of ranges but two choices in one element...

Rate of Change		Delta Change	
Attribute	Value	Attribute	Value
@changeType	changePerSecond	@changeType	changePerSample
@changeBasis	absoluteChange percentageChange	@changeBasis	absoluteChange percentageChange
@spanOfInterestInSamples	ignore	@spanOfInterestInSamples	1 or more
@spanOfInterestInSeconds	1 or more	@spaceOfInterestInSeconds	ignore



Chapter 2 - ParameterType Examples

IntegerParameterType

```
<xtce:IntegerParameterType signed="false" name="CUIAType">  
  <xtce:UnitSet/>  
  <xtce:IntegerDataEncoding sizeInBits="32"/>  
  <xtce:DefaultAlarm>  
    <xtce:StaticAlarmRanges>  
      <xtce:WatchRange minInclusive="-2000.0" maxInclusive="5000.0"/>  
      <xtce:WarningRange minInclusive="-5000.0" maxInclusive="6000.0"/>  
    </xtce:StaticAlarmRanges>  
  </xtce:DefaultAlarm>  
</xtce:IntegerParameterType>
```



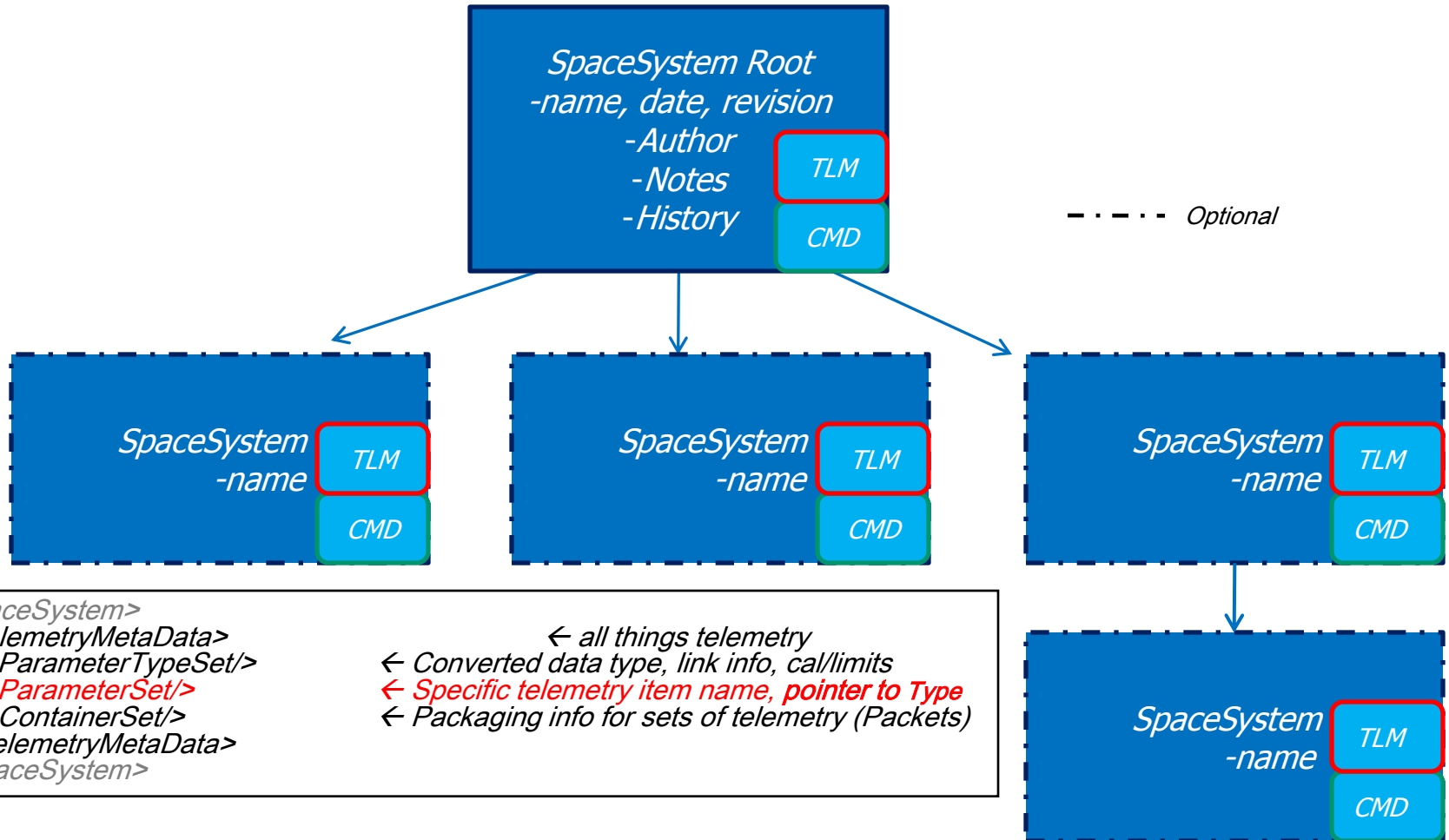
Chapter 2 - ParameterType Examples

FloatParameterType

```
<xtce:FloatParameterType sizeInBits="64" name="LightType">
  <xtce:UnitSet>
    <xtce:Unit description="Bq">Becquerel</xtce:Unit>
  </xtce:UnitSet>
  <xtce:IntegerDataEncoding sizeInBits="16" encoding="twosCompliment">
    <xtce:DefaultCalibrator>
      <xtce:SplineCalibrator>
        <xtce:SplinePoint raw="-32768.0" calibrated="0.0"/>
        <xtce:SplinePoint raw="0.0" calibrated="5.0"/>
        <xtce:SplinePoint raw="32767.0" calibrated="20.0"/>
      </xtce:SplineCalibrator>
    </xtce:DefaultCalibrator>
  </xtce:IntegerDataEncoding>
</xtce:FloatParameterType>
```



Chapter 2 -TelemetryMetaData – Context






Chapter 2 – Parameter: “Points” to a ParameterType

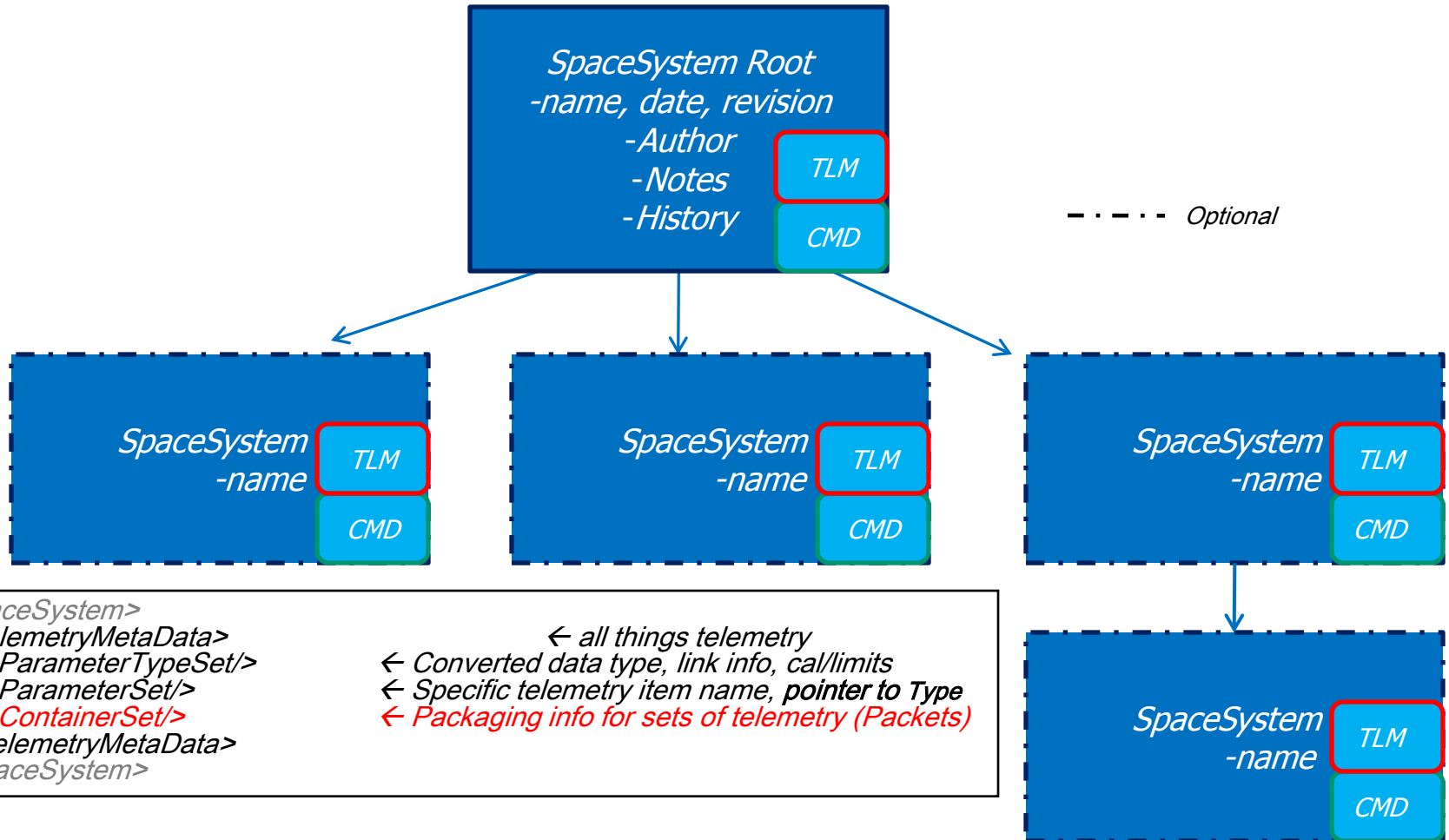
- **Parameter**
 - Holds a name – in your name format “**BAV10089-B**”
 - And a “string pointer” – a Ref – to a ParameterType
 - And an optional “ParameterProperties/@dataSource” attribute
 - Telemetered (default – cmd side ignore)
 - Derived, local, etc...
 - ParameterTypes may be shared by different Parameters...

```
<xtce:ParameterTypeSet>  
    <xtce:IntegerParameterType name="MyParameterType"/>  
</xtce:ParameterTypeSet>  
<xtce:ParameterSet>  
    <xtce:Parameter name="BAV10089-B" parameterTypeRef="MyParameterType"/>  
</xtce:ParameterSet>
```

A red arrow points from the text "MyParameterType" in the parameterTypeRef attribute of the second XML block to the text "MyParameterType" in the name attribute of the first XML block, illustrating the reference relationship.



Chapter 3 -TelemetryMetaData – Context

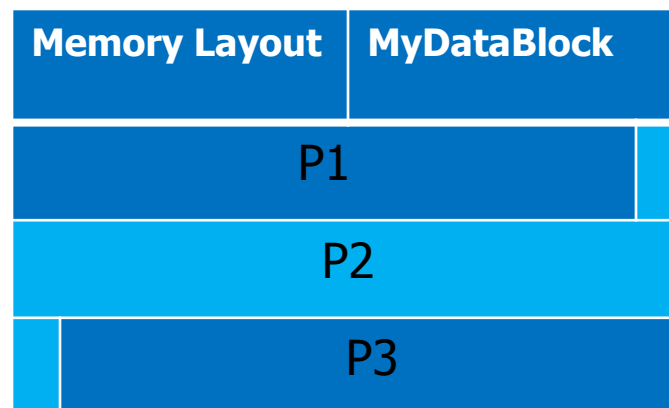




Chapter 3 - XTCE Containers

- Use to represent “memory layout” of Parameters
 - EntryList specifies content and layout:
 - Default: items are “packed” back to back
 - Width taken from ParameterType (look up Parameter, then its Type)
 - Element: `<SequenceContainer>`
 - Very general, may refer to other SeqContainers, Parameters

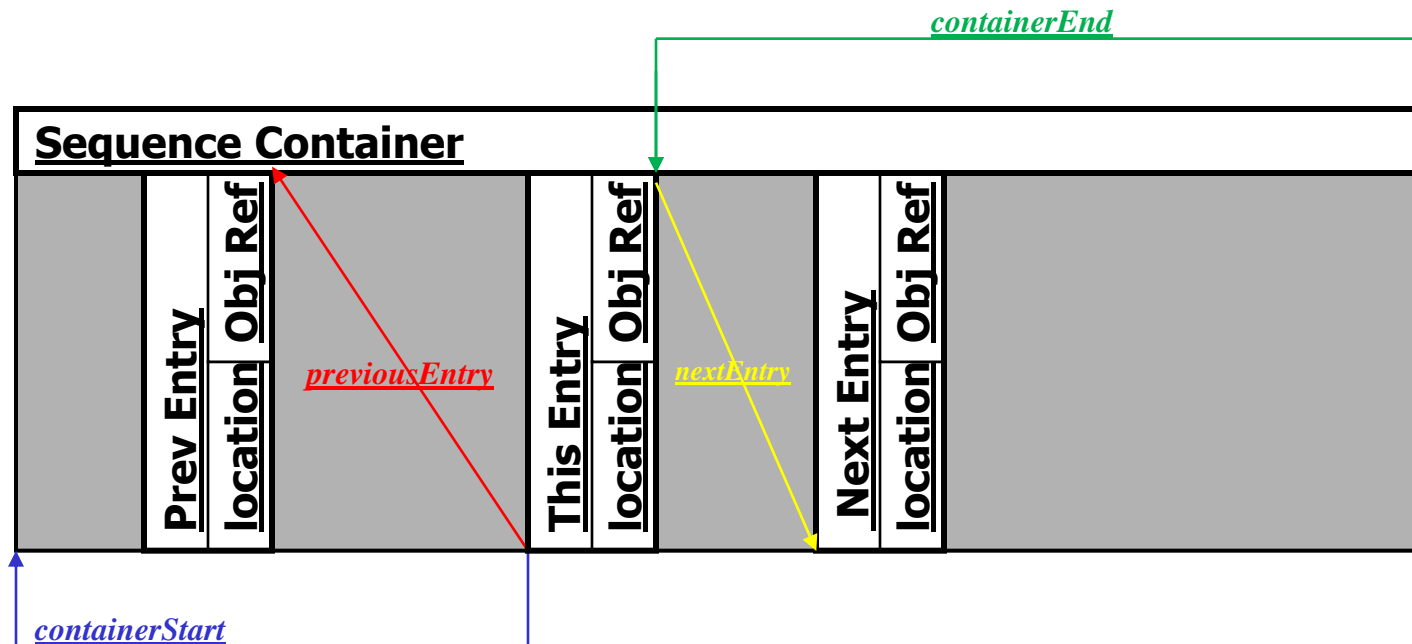
```
<SequenceContainer name="MyDataBlock">  
  <EntryList>  
    <ParameterRefEntry parameterRef="P1"/>  
    <ParameterRefEntry parameterRef="P2"/>  
    <ParameterRefEntry parameterRef="P3"/>  
  </EntryList>  
</SequenceContainer>
```





Chapter 3 - Entry Addressing

- Location of the entry is an integer value referenced from:
 - The end of the previous entry to the start of This Entry (**default**)
 - The start of the container to the start of This Entry (containerStart)
 - The end of the container to the end of This Entry (containerEnd)
 - The start of the next entry to the end? of this entry (nextEntry)





- Use Containers to describe groups of Parameters
 - Headers
 - Packet Bodies
 - Shared groupings of parameters
- EntryList: ParameterRefs, ContainerRefs (others not covered but similar in concept)

```
<xtce:ParameterTypeSet>
  <xtce:IntegerParameterType name="MyParameterType"/>
</xtce:ParameterTypeSet>
<xtce:ParameterSet>
  <xtce:Parameter name="MyParameter" parameterTypeRef="MyParameterType"/>
</xtce:ParameterSet>
<xtce:SequenceContainer name="MyPacket">
  <xtce:EntryList>
    <xtce:ParameterRefEntry parameterRef="MyParameter"/>
  </xtce:EntryList>
</xtce:SequenceContainer>
```

A diagram with two arrows indicating references. A red arrow points from the text "MyParameterType" in the second line of the XML code to the text "MyParameterType" in the fourth line. A blue arrow points from the text "MyParameter" in the fourth line to the text "MyParameter" in the eighth line.



Chapter 3 - Containers – EntryList Options

- The EntryList defines the Parameters in the Container ([parameterRefEntry](#))
 - Each entry has implied back-to-back addressing info
 - Various options like [absolute](#) [addressing](#) can be given
 - The bit-width of each is taken from its ParameterType/DataEncoding size
 - These two together define layout

```
<xtce:EntryList>
  <xtce:ParameterRefEntry parameterRef="Parameter1"> ←
    <xtce:LocationInContainerInBits referenceLocation="containerStart"> ←
      <xtce:FixedValue>0<xtce:FixedValue> ←
    </xtce:LocationInContainerInBits>
  </xtce:ParameterRefEntry>
  <xtce:ParameterRefEntry parameterRef="Parameter2">
    <xtce:LocationInContainerInBits referenceLocation="containerStart">
      <xtce:FixedValue>64<xtce:FixedValue>
    </xtce:LocationInContainerInBits>
  </xtce:ParameterRefEntry>
</xtce:EntryList>
```



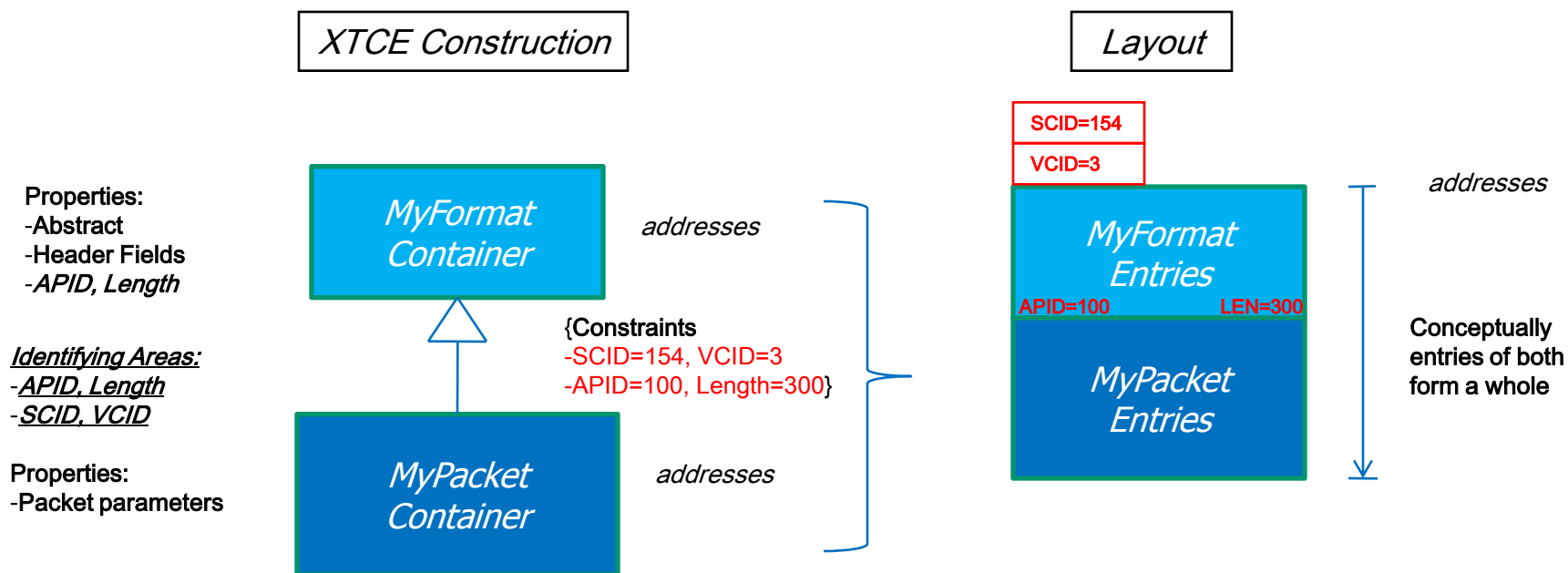
Chapter 3 – Container Inheritance

- Use Container Inheritance to fully describe a packet or minor frame
- Containers may EXTEND another Container (aka “Container Inheritance”)
 - BaseContainer child element
 - Supply the Parent or “Super” Container Ref
 - Supply “Constraints” – a set of conditions that must be true for this description
 - Conditions are usually Parameters in the Parent Container
 - The condition Parameters are probably IDENTIFYING areas of a packet/minor frame:
 - » Ex. CCSDS Land:
 - » APID
 - » Packet Length
 - » SCID, VCID (note: these are not in packet header!)
- Conceptually somewhat similar to Class Inheritance
 - Extending Container gets some things from Parent
 - Principally the EntryList
 - Or it may override some things
 - A few of the other elements in Container (Size)
- Simplified UML Class Diagrams useful to show relationship between containers



Chapter 3 – Basic Telemetry Inheritance

MyPacket



The constraints allow one to match incoming bits with these descriptions – whether literally or conceptually



Chapter 3 -Telemetry Packet Description

MyPacket

The XTCE constructions:

```
<xtce:SequenceContainer name="MyFormat" abstract="true">
  <xtce:EntryList>
    <xtce:ParameterEntryRef parameterRef="APID">
    <xtce:ParameterEntryRef parameterRef="LEN">
  </xtce:EntryList>
</xtce:SequenceContainer>
```

MyFormat IS A SequenceContainer



```
<xtce:SequenceContainer name="MyPacket">
  <xtce:EntryList>
    <xtce:ParameterEntryRef parameterRef="P1">
    <xtce:ParameterEntryRef parameterRef="P2">
    <xtce:ParameterEntryRef parameterRef="P3">
  </xtce:EntryList>
  <xtce:BaseContainer containerRef="MyFormatPacket">
  <xtce:RestrictionCriteria>
    <xtce:ComparisonList>
      <xtce:Comparison parameterRef="APID" value="100"/>
      <xtce:Comparison parameterRef="LEN" value="300"/>
      <xtce:Comparison parameterRef="SCID" value="154"/>
      <xtce:Comparison parameterRef="VCID" value="3"/>
    </xtce:ComparisonList>
  </xtce:RestrictionCriteria>
  </xtce:BaseContainer>
</xtce:SequenceContainer>
```

MyPacket IS A MyFormat

```
<ConceptualEntries>
  <APID/>
  <LEN/>
  <P1/>
  <P2/>
  <P3/>
  <When>
    <APID==100/>
    <LEN==300/>
    <SCID==154/>
    <VCID==3/>
  </When>
</ConceptualEntries>
```

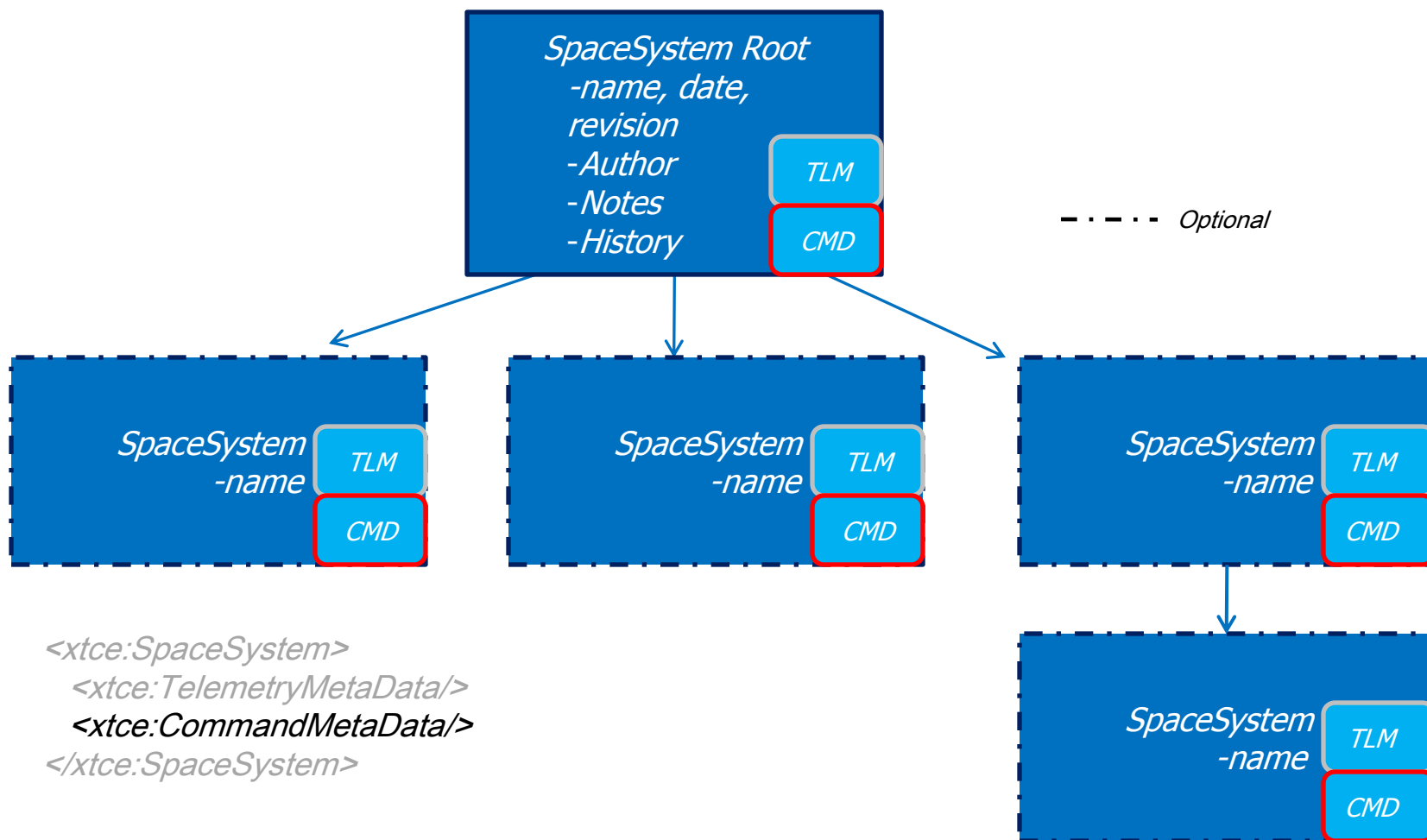
Identifying Constraints

NOTE: SCID and VCID are Session Variables, supplied by the System in this construction – NOT SHOWN



Chapter 4 - CommandMetaData

Child Element of SpaceSystem





Command Descriptions

- XTCE CommandMetaData shares many elements with TelemetryMetaData
 - ParameterSet and ParameterTypeSet are exactly the same
 - ArgumentTypeSet is similar to ParameterTypeSet
 - CommandContainerSet is the same as ContainerSet
- This section then will focus on the differences between the telemetry and command side
 - Most of what has just been presented for telemetry is the same for commanding... but not all of it
 - Some items which of the same construction have a slightly different meaning



```
<xtce:SpaceSystem>  
  <xtce:TelemetryMetaData>  
    <xtce:ParameterTypeSet/>  
    <xtce:ParameterSet/>  
    <xtce:ContainerSet/>  
  </xtce:TelemetryMetaData>  
  <xtce:CommandMetaData>  
    <xtce:ParameterTypeSet/>  
    <xtce:ParameterSet/>  
    <xtce:ArgumentTypeSet/>  
    <xtce:MetaCommandSet/>  
    <xtce:CommandContainerSet/>  
  </xtce:CommandMetaData>  
</xtce:SpaceSystem>
```


Chapter 4 - Command ParameterTypes and ArgumentTypes



- Command ParameterTypes are the same construction as Telemetry ParameterTypes
- ArgumentTypes are similar construction to Telemetry ParmeterType as well
- However the exact relationship of sub-elements is different, as follows:
 - Order is interpreted differently than telemetry – calibrators are “reverse” (aka raw to units)
 - No alarms under ArgumentTypes, they exist under Command ParameterTypes...
 - But probably shouldn't
 - Valid Range may be in one of two places depending on @validRangeAppliesToCalibrated value



ParameterType

- String
- Float
- Integer
- Enumeration
- Binary
- Time
- Array/Aggregate

DataEncoding

- StringDataEncoding
- IntegerDataEncoding
- FloatDataEncoding
- BinaryDataEncoding

- *Generally the various accepted combinations Type/DataEncoding are the same for Telemetry ParameterTypes*
- *ValidRange is slightly different, and no alarms should be specified*

Chapter 4 - CommandMetaData Parameters and Arguments



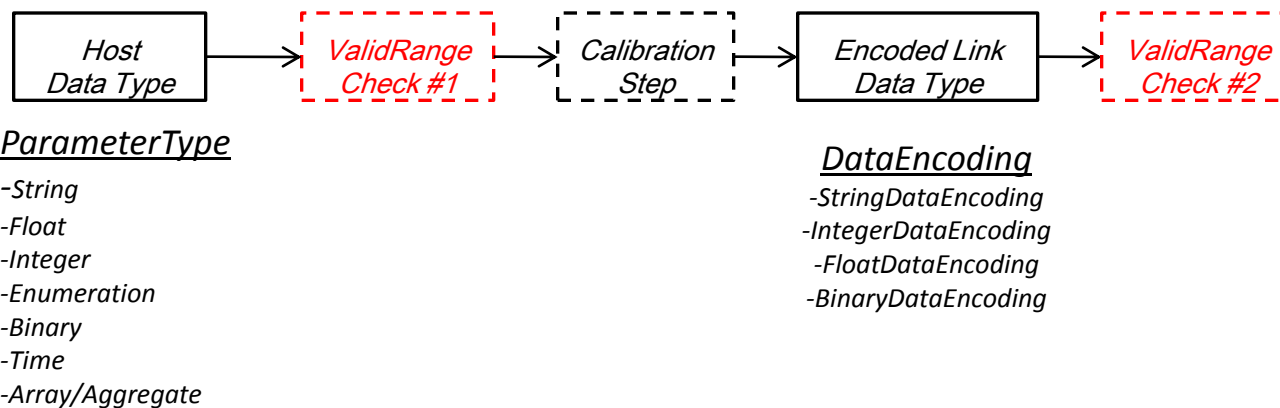
```
<xtce:SpaceSystem>  
  <xtce:TelemetryMetaData>  
    <xtce:ParameterTypeSet/>  
    <xtce:ParameterSet/>  
    <xtce:ContainerSet/>  
  </xtce:TelemetryMetaData>  
  <xtce:CommandMetaData>  
    <xtce:ParameterTypeSet/>  
    <xtce:ParameterSet/>  
    <xtce:ArgumentTypeSet/>  
    <xtce:MetaCommandSet>  
      <xtce:MetaCommand>  
        <xtce:ArgumentList/>  
      </xtce:MetaCommand>  
    </xtce:MetaCommandSet>  
    <xtce:CommandContainerSet/>  
  </xtce:CommandMetaData>  
</xtce:SpaceSystem>
```



- In XTCE Parameters are something set by the System (e.g. a Checksum)
- While an Argument is set the by user
 - Subsystem destination for example, cmd specific
- Arguments are defined local to a Command
 - They are associated with the MetaCommand Element
- Other than this the two are very similar
 - Arguments Ref ArgumentTypes, Parameters Ref ParameterTypes
 - And they are constructed by closely related Schema Types
 - So they have virtually the same elements and attributes within



Chapter 4 - Context Diagram





- ValidRange in Commanding
 - Unlike Telemetry -- can be set to apply BEFORE the command or AFTER the command (but not both)
 - BEFORE
 - leave attribute validRangeAppliesToCalibrated true (default)
 - AFTER
 - set attribute validRangeAppliesToCalibrated to false

Chapter 4 – CommandMetaData Describing Commands



```
<xtce:SpaceSystem>
  <xtce:TelemetryMetaData>
    <xtce:ParameterTypeSet/>
    <xtce:ParameterSet/>
    <xtce:ContainerSet/>
  </xtce:TelemetryMetaData>
  <xtce:CommandMetaData>
    <xtce:ParameterTypeSet/>
    <xtce:ParameterSet/>
    <xtce:ArgumentTypeSet/>
    <xtce:MetaCommandSet>
      <xtce:MetaCommand>
        <xtce:ArgumentList/>
        <xtce:CommandContainer>
      </xtce:MetaCommand>
    </xtce:MetaCommandSet>
    <xtce:CommandContainerSet/>
  </xtce:CommandMetaData>
</xtce:SpaceSystem>
```

MetaCommand/CommandContainer is a LOCAL CommandContainer for Commands

- Build Command Packets here!

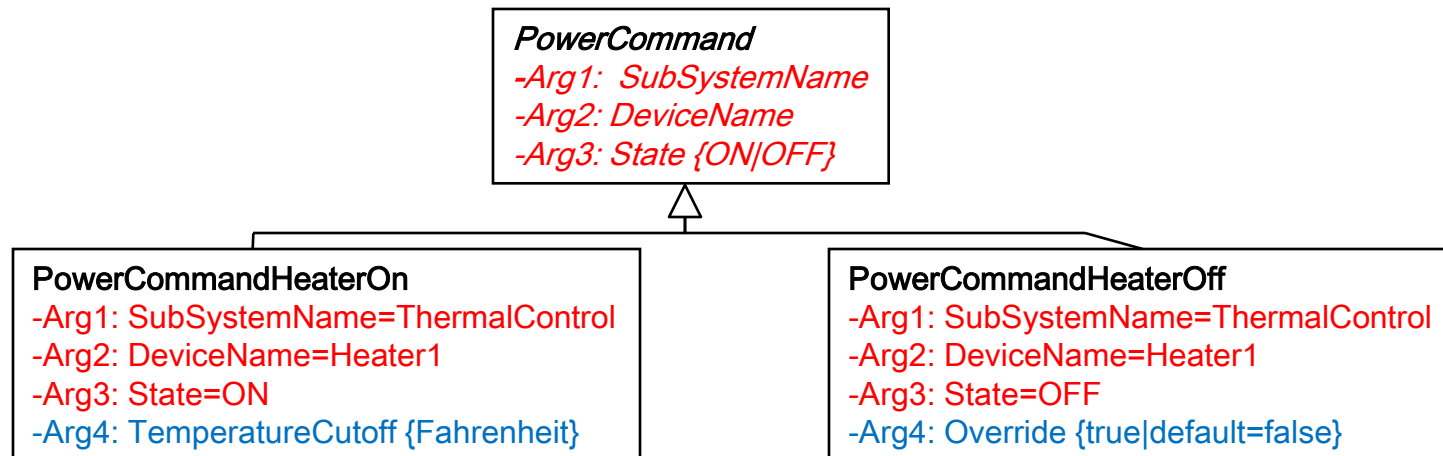


- Use the MetaCommand element to describe Commands
 - Supply Command Arguments
 - A local Container for their Packets
 - Add Verifiers, Interlock, Priority, various constraints, side effects
 - And use **MetaCommand Inheritance** to build up the Command
 - Similar to Container Inheritance
- Use the local MetaCommand/CommandContainer to build its Packet (or Minor Frame)
 - Reference Command Parameters
 - Reference Command Arguments
 - And it may Reference other CommandContainers (CommandContainerSet)
 - Also has Inheritance Mechanism similar to Container Inheritance



- A MetaCommand may EXTEND another
 - One Command may EXTEND another
 - BaseMetaCommand
 - Give the Ref of the MetaCommand being extended
- The extending command can add arguments
 - It gets any Parent arguments and adds its own
- Or it can set arguments in the Parent's Command
 - MetaCommand/BaseMetaCommand/ArgumentAssignmentList

Example



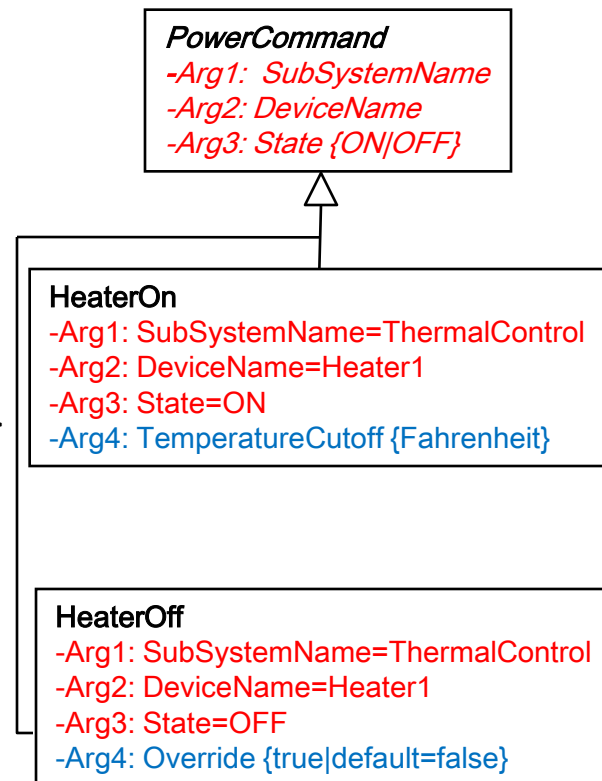


Chapter 4 – Command Inheritance in XTCE

```

<xtce:MetaCommand name="PowerCommand" abstract="true">
  <xtce:ArgumentList>
    <xtce:Argument name="SubSystemName" argumentTypeRef="SubSysEnumType"/>
    <xtce:Argument name="DeviceName" argumentTypeRef="DeviceEnumType"/>
    <xtce:Argument name="State" argumentTypeRef="StateEnumType"/>
  </xtce:ArgumentList>
  <xtce:CommandContainer/>
</xtce:MetaCommand>
<xtce:MetaCommand name="HeaterOn">
  <xtce:ArgumentList>
    <Argument name="TemperatureCutoff" argumentTypeRef="FloatType"/>
  </xtce:ArgumentList>
  <xtce:BaseMetaCommand metaCommandRef="PowerCommand">
    <xtce:ArgumentAssignmentList>
      <xtce:ArgumentAssignment argumentName="SubSystemName" argumentValue="ThermalControl"/>
      <xtce:ArgumentAssignment argumentName="DeviceName" argumentValue="Heater1"/>
      <xtce:ArgumentAssignment argumentName="State" argumentValue="ON"/>
    </xtce:ArgumentAssignmentList>
  </xtce:BaseMetaCommand>
  <xtce:CommandContainer/>
</xtce:MetaCommand>
<xtce:MetaCommand name="HeaterOff">
  <xtce:ArgumentList>
    <Argument name="Override" argumentTypeRef="BooleanType"/>
  </xtce:ArgumentList>
  <xtce:BaseMetaCommand metaCommandRef="PowerCommand">
    <xtce:ArgumentAssignmentList>
      <xtce:ArgumentAssignment argumentName="SubSystemName" argumentValue="ThermalControl"/>
      <xtce:ArgumentAssignment argumentName="DeviceName" argumentValue="Heater1"/>
      <xtce:ArgumentAssignment argumentName="State" argumentValue="OFF"/>
    </xtce:ArgumentAssignmentList>
  </xtce:BaseMetaCommand>
  <xtce:CommandContainer/>
</xtce:MetaCommand>

```



Chapter 4 – MetaCommand's CommandContainer



- MetaCommand has a local CommandContainer
 - MetaCommand/CommandContainer
 - It's similar to the other Containers but DIFFERENT
 - It has a FixedValueEntry to hard code a value
 - It has an ArgumentRefEntry for Arguments
 - It has a BaseContainer but its RestrictionCriteria is optional
 - It has no abstract attribute either!
- Use it to build the Packet or Minor Frame associated with the MetaCommand
 - Any Arguments must be reflected in the EntryList
 - FixedValueEntry might be used to hold opcodes, etc...
 - Any Parameters, that is their values, are supplied by the system (conceptually)
 - And put on the link as is specified in the DataEncoding

MetaCommand's

Chapter 4 – CommandContainer in XTCE



```
<xtce:MetaCommand name="PowerCommand" abstract="true">
```

```
  <xtce:ArgumentList>
```

```
    <xtce:Argument name="SubSystemName" argumentTypeRef="SubSysEnumType"/>
```

```
    <xtce:Argument name="DeviceName" argumentTypeRef="DeviceEnumType"/>
```

```
    <xtce:Argument name="State" argumentTypeRef="StateEnumType"/>
```

```
  </xtce:ArgumentList>
```

```
  <xtce:CommandContainer name="PowerCommandPacket">
```

```
    <xtce:EntryList>
```

```
      <xtce:FixedValueEntry binaryValue="00a0" sizeInBits="16"/>
```

```
      <xtce:ParameterRefEntry parameterRef="Checksum"/>
```

```
      <xtce:ArgumentRefEntry argumentRef="SubSystemName"/>
```

```
      <xtce:ArgumentRefEntry argumentRef="DeviceName"/>
```

```
      <xtce:ArgumentRefEntry argumentRef="State"/>
```

```
    </xtce:EntryList>
```

```
  </xtce:CommandContainer>
```

```
</xtce:MetaCommand>
```

Packet name for example

"OpCode" for example

Check sum, calculated
by the system, inserted here
before uplink

The arguments – order
does not have to match
ArgumentList but they should
all be here...



Chapter 4 - CommandContainer Inheritance

- A MetaCommand/CommandContainer may EXTEND another
 - Another MetaCommand/CommandContainer that is...
 - Similar to Container inheritance in the rest of XTCE
 - But...
 - RestrictionCriteria is optional
- Use it when extending another MetaCommand
 - Must EXPLICITLY set extending MetaCommand/CommandContainer/BaseContainer to the Parent's MetaCommand/CommandContainer
 - (easier to visualize than explain)

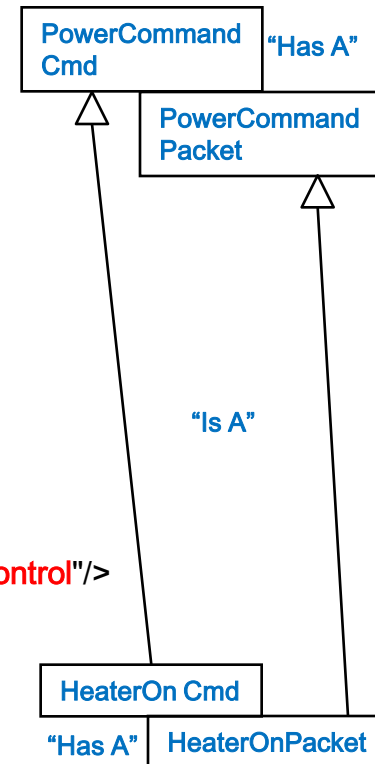
Chapter 4 -

MetaCommand
CommandContainer Inheritance
in XTCE

```

<xtce:MetaCommand name="PowerCommand" abstract="true">
  <xtce:ArgumentList>
    <xtce:Argument name="SubSystemName" argumentTypeRef="SubSysEnumType"/>
    <xtce:Argument name="DeviceName" argumentTypeRef="DeviceEnumType"/>
    <xtce:Argument name="State" argumentTypeRef="StateEnumType"/>
  </xtce:ArgumentList>
  <xtce:CommandContainer name="PowerCommandPacket">
    <xtce:EntryList>
      <xtce:FixedValueEntry binaryValue="00a0" sizeInBits="16"/> <!-- opcode -->
    </xtce:EntryList>
  </xtce:CommandContainer>
</xtce:MetaCommand>
<xtce:MetaCommand name="HeaterOn">
  <xtce:ArgumentList>
    <Argument name="TemperatureCutoff" argumentTypeRef="FloatType"/>
  </xtce:ArgumentList>
  <xtce:BaseMetaCommand metaCommandRef="PowerCommand">
    <xtce:ArgumentAssignmentList>
      <xtce:ArgumentAssignment argumentName="SubSystemName" argumentValue="ThermalControl"/>
      <xtce:ArgumentAssignment argumentName="DeviceName" argumentValue="Heater1"/>
      <xtce:ArgumentAssignment argumentName="State" argumentValue="ON"/>
    </xtce:ArgumentAssignmentList>
  </xtce:BaseMetaCommand>
  <xtce:CommandContainer name="HeaterOnPacket">
    <xtce:EntryList/>
    <xtce:BaseContainer containerRef="PowerCommandPacket"/>
  </xtce:CommandContainer>
</xtce:MetaCommand>

```



MetaCommand

CommandContainer Inheritance in XTCE w/RestrictionCriteria

Chapter 4 -



```

<xtce:MetaCommand name="PowerCommand" abstract="true">
  <xtce:ArgumentList/> <!-- args removed for illustrative purposes -->
  <xtce:CommandContainer name="PowerCommandPacket">
    <xtce:EntryList>
      <xtce:ParameterRefEntry parameterRef="OpCode"/>
      <!-- other entries not shown -->
    </xtce:EntryList>
  </xtce:CommandContainer>
</xtce:MetaCommand>

<xtce:MetaCommand name="HeaterOn">
  <xtce:ArgumentList>
    <Argument name="TemperatureCutoff" argumentTypeRef="FloatType"/>
  </xtce:ArgumentList>
  <xtce:BaseMetaCommand metaCommandRef="PowerCommand">
    <xtce:ArgumentAssignmentList/> <!-- arg assignments removed for illustrative purposes -->
  </xtce:BaseMetaCommand>
  <xtce:CommandContainer name="HeaterOnPacket">
    <xtce:EntryList/> <!-- other entries not shown -->
    <xtce:BaseContainer containerRef="PowerCommandPacket">
      <xtce:RestrictionCriteria>
        <xtce:Comparison parameterRef="OpCode" value="00a0"/>
      </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
  </xtce:CommandContainer>
</xtce:MetaCommand>

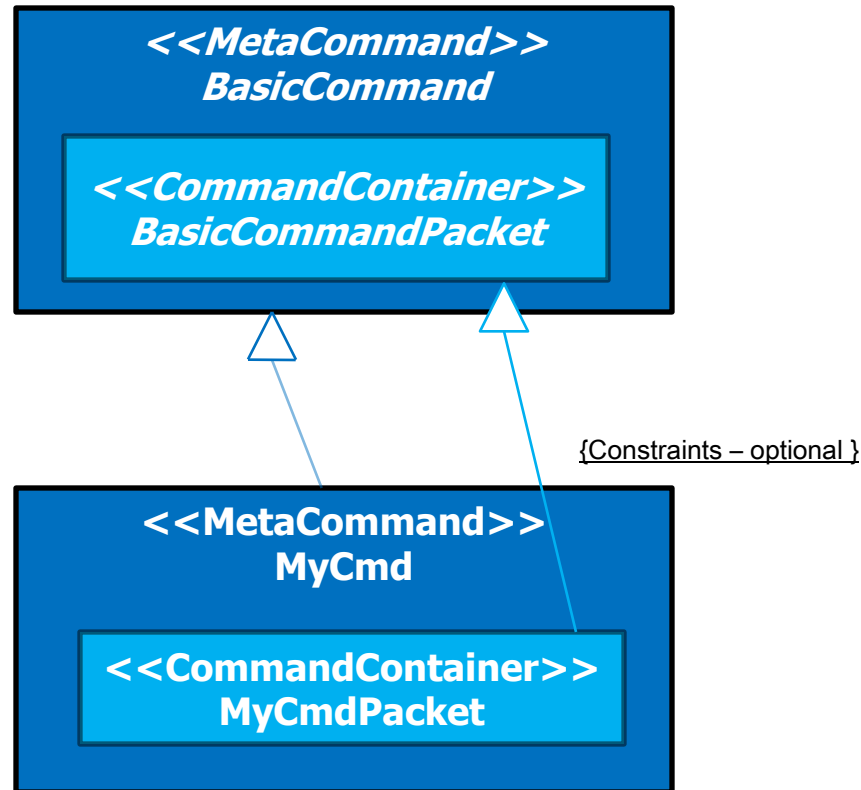
```

System supplies the value for OpCode...

But checks the value here in this constraint...
?Opcode == 0xa0?

Simple constraints of the style "parameter == value" could be automatically processed to essentially inform the system the values that MUST be supplied in the named parameters. You are encouraged to stick with that form for this reason but this is not required.

Chapter 4 –



*A MetaCommand may EXTEND another: MyCmd extends BasicCommand above
Their local CommandContainers extend each other: MyCmdPacket extends BasicCommandPacket*

- *Constraints are optional here*



- XTCE includes these items in MetaCommand:
 - TransmissionConstraint - <TransmissionConstraintList>
 - Check cmd can be run
 - Implied blockage if constraints fail
 - Significance - <DefaultSignificance>, <ContextSignificanceList>
 - Permission/confirmations
 - Interlock
 - Constraints on the next command
 - Verification -- <VerifierSet>
 - Variety of command verification by stages
 - Side Effects – <ParameterToSetList>
 - Side effects after command sent
 - E.g. Command Counter, etc...
 - Suspend Alarms until... <ParameterToSuspendAlarmsSet>
 - Suspend named parameters while cmd takes effect



Chapter 4 – CommandContainerSet

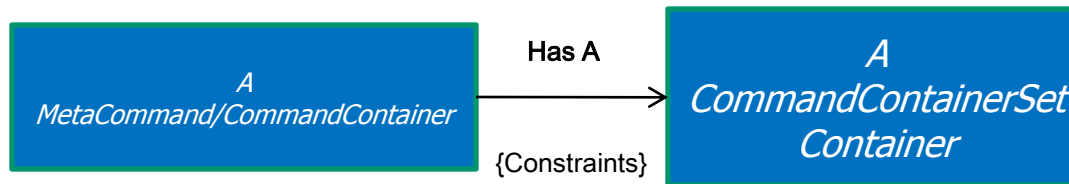
```
<xtce:SpaceSystem>  
  <xtce:TelemetryMetaData>  
    <xtce:ParameterTypeSet/>  
    <xtce:ParameterSet/>  
    <xtce:ContainerSet/>  
  </xtce:TelemetryMetaData>  
  <xtce:CommandMetaData>  
    <xtce:ParameterTypeSet/>  
    <xtce:ParameterSet/>  
    <xtce:ArgumentTypeSet/>  
    <xtce:MetaCommandSet>  
      <xtce:MetaCommand>  
        <xtce:ArgumentList/>  
        <xtce:CommandContainer>  
      </xtce:MetaCommand>  
    </xtce:MetaCommandSet>  
    <xtce:CommandContainerSet/>  
  </xtce:CommandMetaData>  
</xtce:SpaceSystem>
```

CommandContainerSet is NOT the same thing as MetaCommand/CommandContainer



Chapter 4 – CommandContainerSet

- Built using the same XTCE Schema-types as ContainerSet in TelemetryMetaData...
- Generally meant to be used as the “pieces/parts” of MetaCommand/CommandContainers
 - Chunks of command parameters that repeat or reused by various commands
- A “has a” relationship



*For example – perhaps a Command Packet has an optional secondary header
- Constraint: Secondary Header Flag must be 1*

Chapter 5 – The Forgotten Elements



- Lesser used major elements include:
 - TelemetryMetaData/MessageSet
 - StreamSet
 - AlgorithmSet
 - ServiceSet



- Deprecated...
- Essentially an alternative way to build a Packet or Minor Frame...
 - A purely "HAS A" relationship to Containers
 - Retained from an earlier version of XTCE



Chapter 5 - StreamSet – Streams

- A set of elements to describe aspects of “bit streams”
 - Possibly could be used for portions of frame description, etc...
- May be included in containers directly
 - Perhaps a special “sub-stream” in a container
- CCSDS – not quite enough elements to FULLY describe the “CCSDS stack” from the frame-sync to packets
 - Lacks RS encoding info for example
 - Generally focus on packet descriptions and leave the FEP for others
- Links to a container
 - Probably a “super container” representing the frame...
 - w/derived child container representing specific packets
- Or links to a service
 - Services are abstractions, perhaps a service has certain streams within it...



- Used to describe algorithms used in your tlm/cmd processing
 - Many options – from actually including code text, to simply the name of a function...
- Two forms:
 - Custom: functions, methods, procedures...
 - Math: equations using postfix notation...



- An abstraction – groups “logically like” Containers
- For example:
 - Suppose your mission splits its functionality across multiple CCSDS Virtual Channels
 - Each Channel may represent a “service”
 - A Service element could be defined and the list of Containers for each APID in the service (VCID) added
 - Options:
 - point to the “super container” per channel
 - Point to each “final” packet container
 - Point to all the containers making up packets on those channels...
 - Etc...?
 - Totally user defined...



Chapter 6 – Implementing XTCE

- A number of ways to implement XTCE
 - #1 - XSLTs, XPath, etc...
 - #2 - DOM or SAX custom parser
 - #3 - Data-type mapper (XTCE Schema Type to Programming language Classes – “Autogenerators”)
 - #4 - Combination
- Focus on #3
 - Numerous “autogenerators”
 - Eclipse IDE EMF (Java used)
 - XMLSpy Enterprise (built-in, Java: used)
 - JAXB – now part of Java distro (used)
 - Others: XMLBeans, Castor, ..., many others (& other languages)
 - Items to Look For:
 - The order of certain XTCE elements is important
 - Native support or easy bail out to low level processing
 - Ex. Entry order means something
 - Comments or other schema items may not be available directly in mapping
 - Method for determining if attribute value is from a default value or set directly in XML
 - It would be nice but general support seems lacking:
 - DOM \leftrightarrow Your Mapped Classes
 - Reason: other XML APIs may sit on DOM
 - Some (minimal) support in JAXB
 - Old XMLSpy mapper sat on top of DOM, not sure of new ones
 - EMF uses its own ECore...



Chapter 6 – A Quick Look at JAXB

- JAXB – Java Architecture for XML Binding
 - Will map your XML Schema (i.e. XTCE) to Java Classes
 - Has additional features including “binding” configuration file
- Used to implement most of GSFC XTCE prototype software
 - Plusses:
 - Java Generics and Collections well supported
 - Mapping tunable w/config file
 - Now part of official distribution
 - Everyone has it that downloads Java
 - Some support for DOM nodes
 - But not built on DOM (that I can tell!)
 - ...?
 - Misses:
 - Can’t seem to parse or build comments
 - Can’t convert any unmarshalled object to a DOM node
 - But seem to be able convert any DOM node to a marshalled object
 - ...?



Chapter 6 - JAXB SchemaTypes to Java Mapping

XML Schema DataType → Mapping to Java Type or Class
(some)

Examples

```
xsd:string → java.lang.String
xsd:integer → java.math.BigInteger
...
xsd:dateTime → javax.xml.datatype.XMLGregorianCalendar
...
xsd:double → double

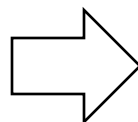
Etc...
```



Chapter 6 - JAXB Ex. XTCE SchemaType to Java

XTCE SpaceSystemType

```
<complexType name="SpaceSystemType" mixed="false">
<complexContent mixed="false">
  <extension base="xtce:NameDescriptionType">
    <sequence>
      <element name="Header" type="xtce:HeaderType"
        minOccurs="0"/>
      <element name="TelemetryMetaData"
        type="xtce:TelemetryMetaData" minOccurs="0"/>
      <element name="CommandMetaData"
        type="xtce:CommandMetaData" minOccurs="0"/>
      <element name="ServiceSet" minOccurs="0">
        <complexType>
          <sequence>
            <element name="Service" type="xtce:ServiceType"
              maxOccurs="unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element ref="xtce:SpaceSystem"
        minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="operationalStatus"
      type="token" use="optional"/>
  </extension>
</complexContent>
</complexType>
```



XTCE SpaceSystemType Java Class

```
public class SpaceSystemType extends NameDescriptionType
{
  protected HeaderType header;
  protected TelemetryMetaData telemtryMetaData;
  protected CommandMetaData commandMetaData;
  protected SpaceSystemType.ServiceSet serviceSet;
  protected List<SpaceSystemType> spaceSystem;
  protected String operationalStatus;
  public HeaderType getHeader() { return header;}
  public void setHeader(HeaderType value) { this.header = value;}
  public boolean isSetHeader() { return (this.header!= null);}

  public TelemetryMetaData getTelemetryMetaData() {
    return telemtryMetaData;
  }
  public void setTelemetryMetaData(TelemetryMetaData value) {
    this.telemetryMetaData = value;
  }
  public boolean isSetTelemetryMetaData() {
    return (this.telemetryMetaData!= null);
  }
  public CommandMetaData getCommandMetaData() {
    return commandMetaData;
  }
  public void setCommandMetaData(CommandMetaData value) {
    this.commandMetaData = value;
  }
  public boolean isSetCommandMetaData() {
    return (this.commandMetaData!= null);
  }
}

// ... DELETED FOR SPACE REASONS
}
```

Note: items edited to fit....



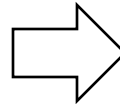
Chapter 6 – Simple JAXB XTCE Example

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.Marshaller;
import org.omg.space.xtce.ObjectFactory;
import org.omg.space.xtce.SpaceSystemType;
```

```
public class SimpleSpaceSystem {

    public static void main(String[] args) {
        JAXBElement<SpaceSystemType> spaceRoot;
        JAXBContext jc;
        Marshaller marshaller;
        ObjectFactory factory;

        try {
            jc = JAXBContext.newInstance("org.omg.space.xtce");
            marshaller = jc.createMarshaller();
            marshaller.
                setProperty(
                    "com.sun.xml.internal.bind.namespacePrefixMapper",
                    new XTCEPrefixMapper());
            marshaller.
                setProperty(Marshaller.JAXB_SCHEMA_LOCATION,
                    "http://www.omg.org/space/xtce SpaceSystemV1.1.xsd");
            marshaller.
                setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
                    new Boolean(true));
            factory = new ObjectFactory();
            SpaceSystemType space = factory.createSpaceSystemType();
            space.setName("HelloWorld");
            spaceRoot = factory.createSpaceSystem(space);
            marshaller.marshal(spaceRoot, System.out);
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xtce:SpaceSystem xmlns:xtce="http://www.omg.org/space/xtce"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  name="HelloWorld"
  xsi:schemaLocation=
    "http://www.omg.org/space/xtce SpaceSystemV1.1.xsd"/>
```

Note: XTCEPrefixMapper not shown



Chapter 6 – Harder XTCE Items to Implement

- General NameReferences and NameReference Instances
 - All “refs” should point to something valid: you must implement this
 - Absolute addresses: /a/b/c and various forms of relative addresses: ../b/c, a/../b, etc...
 - “Path-less” nameRefs which may involve a partial tree search: c
 - Instances: Adds array syntax: /c[1], And aggregate syntax: /a/b/c.subField
 - And relates to some sort of “instance table”
- Container Inheritance, EntryList, IncludeConditions, etc...
 - Various general aspects of inheritance, EntryList addressing
 - Multiple levels, containerRefs that themselves have inheritance, and so on
- XTCE Type Inheritance
- Dynamic Container Match & NextContainer
 - Never been implemented yet that we are aware of...

(a few more not mentioned)

But wait, I have to implement all of those? Depends...



Chapter 6 – Native Implementation

- An implementation that uses XTCE internal to a toolchain
 - Gets packet stream on input, uses XTCE to pull-apart into host-side telemetry items
 - Or build cmds for uplink using definitions to supply args, values & format
- High degree of automation or self-configuration possible
 - Possibly implement every feature of XTCE
 - Thus potentially ingesting ANY XTCE file that is correct to self-configure software for any incoming data stream
 - But of course one can always constrain or restrict XTCE support if necessary
- Implementations?
 - Some aspects of GSFC prototype software falls into this category...
 - But never used to actually decommutate binary packet data (yet)
 - University of Bremen (Germany) implementation used XTCE to configure itself for decommutating binary packet file
 - Demo'd ... but source?



- 100% XTCE Exchange between teams using same features
 - Agreement ahead of time on which features of XTCE to use
 - Depends somewhat on hardware of team members
 - If controlled or standardized between members: 100% exchange possible
- As these agreement don't exist, 100% exchange is not possible without COMMON AGREEMENTS
 - Ex. Extreme case: I support only telemetry, you support only Commands. Our XTCE files are 100% incompatible!
 - But even differing formats may offer partial exchange
 - NASA CCSDS & ESA PUS Mission were able to exchange the majority of telemetry metadata even though basic packet formats ultimately differ... but it took some work
- Implementations?
 - GSFC JWST Translator
 - CNES "Best" Suite
 - ESA Cryosat...
 - Several others... mostly on the "Customer" side – (NASAs of the world)

Chapter 6 – Ingredients in Successful Exchange



- Now:
 - Determine exchangees (team members), agree on common “flavor” of XTCE features
- Future:
 - Industry agreed upon flavors, pick the category you fall into...
- Develop common XTCE usage “constraints” to format
 - CCSDS is a “format” but stops at the header
 - You have to get down deeper Examples:
 - Polynomials, # of coefficients
 - Alarm levels
 - Floating point sizes and formats
 - Bit/byte ordering...
 - Etc..., etc..., etc...
- Implement to these constraints & 100% exchange is possible between team members, institutions, and so forth

Chapter 6 – Automating Exchange?



- Write XTCE constraints in a computer processable manner
- Process constraints to configure validation software
 - Use in conjunction with XML Parsing to validate XML in “your” XTCE
- Process constraints to configure “XTCE builder” software
 - An API that only allows XTCE files to be constructed in “your” flavor
- Are all constraints describable in a computer processable manner?
???
- Is this an additional area of standardization?

Summary



- XTCE is a large schema with many advanced features for most tlm/cmd systems
 - XML, Object Oriented Model for “packaging descriptions” and Commands
- Native Implementers may implement every facet of XTCE
 - Self-configuration and automation may be high
- Exchange Implementers may contend with “flavors”
 - Ensure team members are on the same page: common XTCE constraints
 - Some level of automation may be possible, process constraints automatically
 - Generate “constraint validation” and “builder” tool
- Move towards industry standard flavors



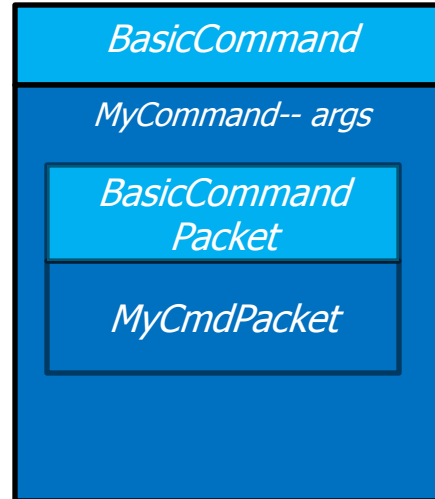
~Backup Slides~



Chapter 4 – We want to build this

The Command:

- *An abstraction for “people”, includes arguments for them to set*
- *Its Packet, the real bits that make up the command for uplink*

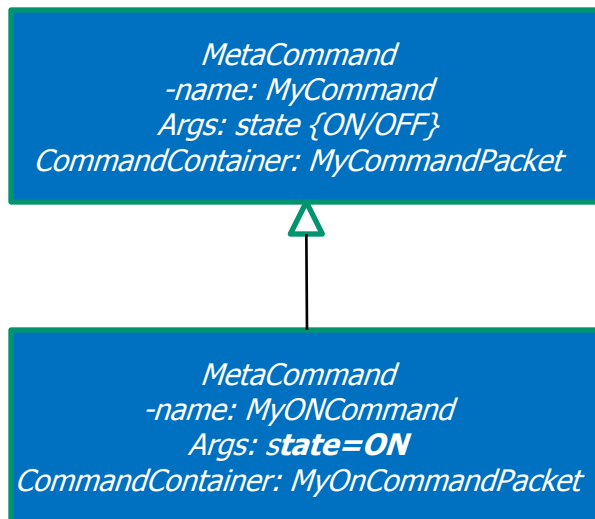


Although the command itself is an abstraction, its packet is a bit-packed construct

- *That's inner box in the diagram here*



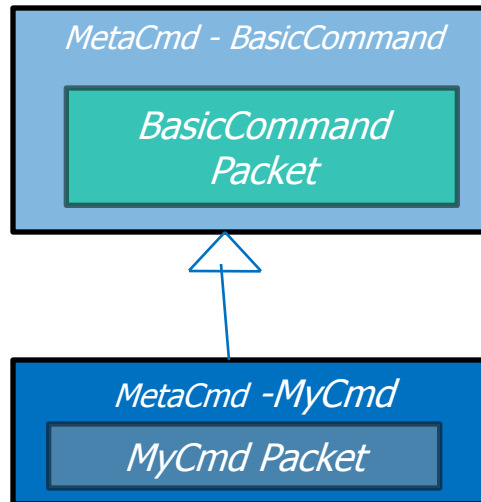
- Arguments – arguments to the command
 - MetaCommand/ArgumentList element
- CommandContainer – private container for describing the command packaging (but can “see” ContainerSet and CommandContainerSet)
 - MetaCommand/CommandContainer element
- Inheritance – one metacommmand may extend (“sub-class”, derive) another
 - BaseMeta element



Sets State to ON - MetaCmd Inheritance,
Child can set Parent's
arguments

Not Shown: private MyOnCommandPacket Container also inherits from MyCommandPacket Container

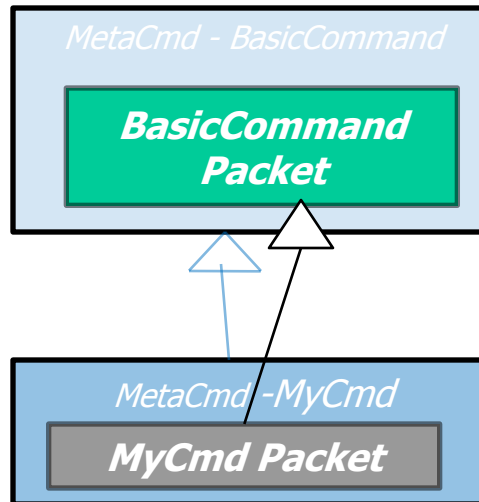
Chapter 4 – Basic Command Packet Pattern



Note: MyCmd Packet EXPLICITLY extend BasicCommand Packet in XTCE, NOT SHOWN

Chapter 4 –

Private MetaCommand/CommandContainer
Also Inherits



*MyCmd Packet EXTENDS BasicCommand Packet - link must be **explicitly** set*
*-Constraints are **optional***
-Constraint are probably Identifying area of Command Packet: values inserted into stream



Chapter 4 - MetaCommand Inheritance

The MetaCommand Part

```
<xtce:MetaCommand name="BasicCommand" abstract="true">
  <xtce:CommandContainer name="BasicCommandPacket"/>
</xtce:MetaCommand>
<xtce:MetaCommand name="MyCmd">
  <xtce:BaseMetaCommand metaCommandRef="BasicCommand"/>
  <xtce:CommandContainer name="MyCommandPacket"/>
</xtce:MetaCommand>
```

"BasicCommand IS A MetaCommand"

"MyCmd IS A BasicCommand"

Note: args/details not shown

The MetaCommand /CommandContainer Part

"BasicCommandPacket IS A MetaCommand/CommandContainer"

"MyCommandPacket IS A BasicCommandPacket"

Note: details left out for illustrative purposes



```
<xtce:MetaCommand name=" BasicCommand " abstract="true">
  <xtce:CommandContainer name="BasicCommandPacket"/>
</xtce:MetaCommand>
<xtce:MetaCommand name=" MyCmd ">
  <xtce:BaseMetaCommand metaCommandRef=" BasicCommand ">
    <xtce:CommandContainer name="MyCommandPacket"/>
    <xtce:BaseContainer containerRef="BasicCommandPacket"/>
  </xtce:CommandContainer>
</xtce:MetaCommand>
```

"MyCommandPacket IS A BasicCommandPacket"

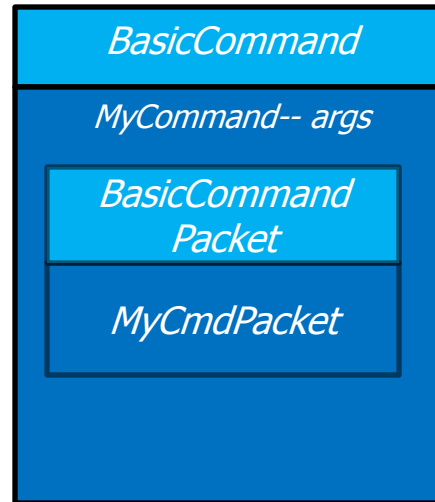
"BasicCommandPacket IS A MetaCommand/CommandConainer"

*Note: Identifying Constraints not shown, similar to Telemetry, but semantics are values are **inserted** into stream*



Chapter 4 – Command & Packet

What we got



Although the command itself is an abstraction, its packet is a bit-packed construct